Sustainable AI

Evaluating the energy efficiency of PyTorch and TensorFlow frameworks

Computer System Performance
KSCOSYP1KU

Wednesday 17th May, 2023

Claudio Sotillos Peceroso
clap@itu.dk

Ívar Óli Sigursson
ivas@itu.dk

Jorge del Pozo Lérida
jord@itu.dk

I. Introduction

Deep learning frameworks, such as PyTorch and TensorFlow, have revolutionized the field of artificial intelligence (AI) by facilitating the development of complex artificial neural networks for solving tasks like image classification, object detection, and natural language processing. As the demand for deep neural networks applications continues to grow, optimizing the performance and energy efficiency of these frameworks on high-performance computing (HPC) clusters has become a critical research area.

In this project, we aim to investigate the power consumption of PyTorch and TensorFlow frameworks using the popular ResNet (Residual Neural Network) architecture [1]. ResNet is a deep neural network architecture that has shown remarkable performance in various image classification tasks. However, little is known about the power consumption characteristics of these frameworks when used with ResNet models of increasing complexity, as well as with varying workload complexities.

Understanding the power consumption of deep learning frameworks is crucial for optimizing energy efficiency and reducing operational costs in large-scale AI deployments. By measuring the power consumption of PyTorch and TensorFlow on an HPC cluster, and analyzing the impact of increasing neural network and workload complexities, we can gain insights into the energy efficiency of these frameworks and inform decisions on resource allocation and system design.

Furthermore, this project has significant implications for sustainability and environmental concerns. As energy consumption in data centers continues to rise, it becomes imperative to study and mitigate the environmental impact of deep learning frameworks. Our findings can contribute to developing more energy-efficient AI systems and help in making informed decisions towards building sustainable AI infrastructure.

II. Experimental methodology

We will now proceed to describe our experimental setup, indentifying the different steps of the experimental design process:

- ***Goal:*** compare the power consumption of most used deep learning frameworks *TensorFlow* **VS** *PyTorch* when learning to solve a simple image classification task. Therefore, exactly the same experiments and analysis are done for each framework.
- ***System:*** our experiments will run on ITU's HPC cluster. We will be utilizing the compute node *desktop4*, which contains a single *RTX 2070* GPU and an 8-core CPU with 32 GiB memory.
- ***Metric:*** power consumption in Watts, energy used in Jules, GPU utilization, GPU temperature and test accuracy. We will measure these metrics using Nvidia's *nvidia-smi*, which is a CLI that provides monitoring capabilities for GPUs with an Nvidia driver.
- ***Parameters (k):*** the parameters we will take into account are the complexity of the ResNet, the complexity of the classification problem, and the batch size.
- ***Levels (l):***
  - Regarding the complexity of the ResNet model, this will be achieved by using the predefined Resnets architectures from both frameworks. The available architectures are ResNet50, ResNet101, and ResNet152.
  - The complexity of the classification problem will be achieved with training on different datasets. These

sets are the *CIFAR10* (50.000 32x32 images with 10 classes), *CIFAR100* (50.000 32x32 images with 100 classes) datasets [2], and the *SVHN* (600,000 32x32 images with 10 classes) dataset [3].
- For the Batch size, in order to see the impact of increasing the clusters of images used at each epoch of training, we have chosen the most commonly used batch sizes when training this type of CNNs for this task (32, 64,128, and 256).

- Type of Experiment: mainly analytical since in order to perform the measurements of our metrics we have used "nvidia-smi".

- Workload: image classification model training on the defined datasets. The total number of epochs is fixed and of size 10. It is important to clarify that even if we focus on the energy efficiency analysis, we also evaluate how well each model is doing at the end of each epoch to verify that they are actually learning.

- Experimental Runs: for each unique combination of levels, we make two different runs and average between them. Because of the repetitive nature of deep learning, we could also consider each epoch as a separate run, as it is known that behaviour between epochs do not change much (except for first epochs, where all startup processes take place).

A. Summary

- **Goal:** *Tensorflow* **VS** *PyTorch*
- **System:** ITU HPC node *desktop4*
- **Metric:** Power consumption in watts, total power consumption, and GPU utilization
- **Parameters(k):** Model complexity, classification complexity and batch size
- **Levels(l):** Framework, Data Set, Resnet Architecture, Batch Size
- **Type of experiment:** Measurements mainly
- **Workloads:** Synthetic Workload (benchmarking)
- **Experimental Runs:** 2 runs

Other relevant information regarding the HPC compute node and the implementation details is shown in Table I. The complete documentation on the ITU's HPC cluster can be found here (note: need to be connected to ITU's network for access).

Cores (Threads/core)	8
RAM	32 GiB
GPU	1x RTX 2070
Operating System	Ubuntu 18.04.6 LTS
Programming Language	Python
Tmp space	400 GiB
Interconnect	1 Gbps Eth

TABLE I: Specifications of our system

B. Experiment Development

All experiments carried of for each unique combination of levels are done both for PyTorch and for TensorFlow. Because the syntax for training default ResNets imported from each framework varies, there are two separate scripts for executing the experiments, one per framework. They both share some utility functions imported from another script. All code and results obtained throughout the project can be found in ComputerSystemsPerformance [4] GitHub repository. The step-by-step workflow followed to carry out the experiments will be explained hereafter.

To begin with, we started by making sure that both frameworks used exactly the same data for training. This is important because when training models in these frameworks, one normally loads them using their specific modules for such purpose. Thus, to eliminate divergences in possible data pre-processing for each framework's custom module, as well as for making sure that the same images are used for train test splits, we loaded the datasets from *torchvision* package for both frameworks. Therefore, functions used to load the datasets are shared between framework scripts to achieve this goal.

Then, another thing to take into account is how the loading of the default model. To guarantee that both models start learning from the same initial weights, we do not load them pre-trained. This way, we ensure that the boundary conditions for the weights optimization is exactly the same (otherwise, models could take different path in the gradient descent due to different start point).

Once data and model loading is handled for each script, the training part part comes. The way of training each framework differs a bit, but on the high level the idea is the following: optimize for each batch and epoch, and at the end of each epoch, evaluate the model on the test set and insert output (accuracy, recall, precision, f1, timestamp) to a file.

Python script for training each framework were designed to be run on the command line with level values as input argument. For the sake of running all experiments on the HPC, we created one bash script for each framework that looped over all combination of level values and executed the following command sequentially:

1) start recording with nvidia-smi into a file in the background
2) execute Python training script for level combination
3) kill first process when Python script finishes

So, at each iteration two files are created, one from model training script and another one for energy data from nvidia-smi output (timestamp, power.draw, temperature.gpu, memory.used, utilization.gpu, utilization.memory). Therefore, we get two time-series csv files for each experiment (see Figure 1, where energy data saved for a single run is plotted through time). Both were saved with a consistent filename pattern to facilitate later post-processing and analysis of results.

It must be mentioned that running the experiments on the HPC with GPUs was very hard (especially with Tensorflow) and time-consuming task, due to very strict version requirements of other software (python version, CUDA version, framework version, other complementary libraries, etc) having to use different environments and resolve thousands of compatibility issues. It must also be noted that we did some initial experiment to determine if it was worth it to include more epochs, but we observed in the results that increasing the number of epochs wouldn't make a significant change, so we thought that using more levels of batch size would give more relevant results.

After making it work on both frameworks and making sure that they were using correctly the GPU while training, we scheduled the jobs in the HPC and obtained the results in CSVs. Therefore, the last and also time-consuming task was to process the output data.

The processing of the data went as follows. First, individual experiments were inspected as seen in Figure 1. For each experiment run data was grouped average by epoch: all data gathered for the time frame corresponding to some epoch was averaged. This allowed to easily merge with model scripts output data, which contains data about evaluation metrics. Then, the values obtained were averaged between different experimental runs (it would not be possible otherwise, as epochs could take slightly different time for each run and it would be impossible to merge correctly). Energy used at epoch was obtained by multiplying total time taken for the epoch to finish by the average power through that epoch. Lastly, other aggregation calculations were made.

Finally, it is worth clarifying that a single experiment consists in training a certain Resnet architecture (Resnet50/101/152) of a framework (PyTorch or TensorFlow) on one of the datasets (CIFAR10, CIFAR100, or SVHN) using a concrete batch size (32, 64, 128, or 256) during 10 epochs. As mentioned, each experiment is run twice (2 runs) and the average of these two runs is computed. Figure 1 shows the result of one of the experiment runs over time for both Frameworks.

III. Interpretation of Results

In Figure 2 we are able to see the total energy used for a 10 epoch training for all combinations of levels.

The first thing we can see from all the graphs is that clearly the use of larger batch sizes has a significant impact on energy consumption. When smaller batch sizes are used, model parameter updates are more frequent. These updates can be costly in both time and energy, especially if the network architecture is complex or the training data is large (we can also see how training with larger datasets requires more energy). Using larger batches reduces the frequency of parameter updates, which in turn reduces communication overhead. In addition, GPUs are optimized to process large amounts of data in parallel. Using larger batch sizes allows the hardware to utilize its parallel processing capability better, as the computations required to process each batch can run concurrently. This can result in faster training times and lower energy consumption per batch.

Furthermore, we can observe how models with more layers need much more energy to be trained. As the number of layers in a ResNet increases, so does the computations needed to process each input and the memory requirements for storing intermediate results during training. This means that more energy is needed to perform the necessary calculations and memory operations, which can prolong training time and increase energy consumption.

Ultimately, notice how in each case Tensorflow takes more (or the same) energy for training its architectures. We don't know exactly why TensorFlow always takes so much energy to train. We think it might be because TensorFlow and PyTorch might use different GPU utilization strategies, which could impact energy usage. Also, data has to be accessed quite often, and frameworks use different methods for data loading and preprocessing (the data is the same but Pytorch uses an object called "dataloader" to iterate over the dataset using the batch size specified, whereas TensorFlow takes the data directly into its fit function), which could impact energy usage during training. Additionally, differences in the implementation of the forward and backward passes

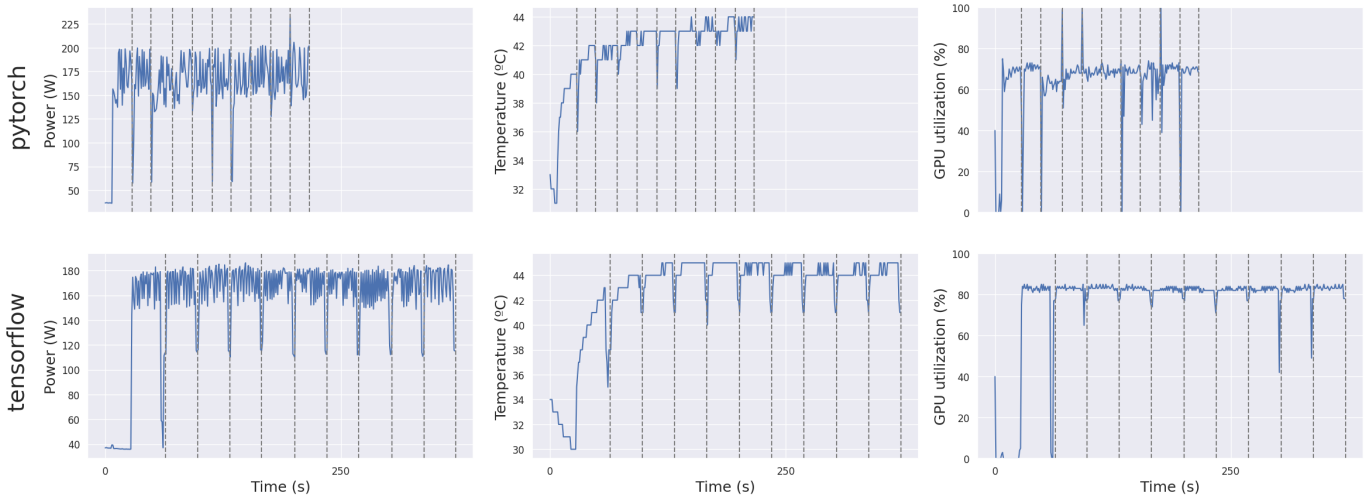

Fig. 1: Single experiment run for both frameworks through time. Levels: $batch\_size = 128, dataset = SVHN, model = resnet101$. Vertical dashed lines indicate the end of every epoch

through the ResNet architecture in each framework could lead to different energy usage.

If we look at Figure 3, it represents the average power consumption (measured in Watts = Joules per second) in each of the scenarios. As can be seen, Tensorflow consumes more power on average, but when the batch size is very large it seems that PyTorch isn´t as well optimized for working with such large batch sizes (maybe because the batch size of 254 isn´t as common as the previous ones). It should be noted that we have plotted epoch 1 separately and then the rest of the epochs. In this first epoch Pytorch has a higher average power consumption than Tensorflow in most scenarios, so it can be concluded that Tensorflow is more efficient at the beginning but takes more average power on average in the long run.

These first Figures give us a general overview of what is happening with the energy consumption in each scenario. Nonetheless, we should also examine the time series obtained in a single experiment to understand properly what is happening during training.

Figure 1 represents this, and although we are only showing the results of a single experiment we have plotted the rest and they are more or less similar. The only thing that varies is that in some scenarios PyTorch finishes training long before and in others, they finish more or less at the same time. During the first epoch, the energy usage is low for both frameworks until it spikes up and stays roughly at the same level until the training has finished (this is also a common aspect in the rest of the experiments, not just in this one). This is most likely because we are loading the data in memory and sending it to the GPU, so the GPU stays idle in the meantime. We also notice in Figures 1 and 3 that PyTorch finishes way earlier than Tensorflow, while only having slightly higher average energy usage. This hints towards PyTorch being more efficient with its GPU usage. We can confirm that by looking at Figure 4 where we see that the average GPU utilization is lower for Pytorch, which might mean that it's more efficient with the utilization than Tensorflow.

Finally, if we look at Figure 5, we can see how much the models have learned in each of the scenarios. The results are not brilliant in terms of accuracy of classification, but this is totally normal since we use the same parameters for each dataset, and for such complex networks to learn properly the different features of each dataset, their hyperparameters have to be modified repeatedly until finding the most appropriate combination or even increase the number of epochs. We can observe that the results are quite even for both frameworks. It is noticeable how poorly the models perform with the CIFAR100 dataset. We believe that this is due to the fact that it has a very large number of classes compared to the other two datasets and that to obtain better results it would be necessary to train the models for more epochs (and possibly do some data augmentation since we now have fewer images per class compared to CIFAR10).

In summary, PyTorch demonstrates faster and more efficient learning capabilities compared to TensorFlow, besides consuming less energy in all the experimental scenarios.

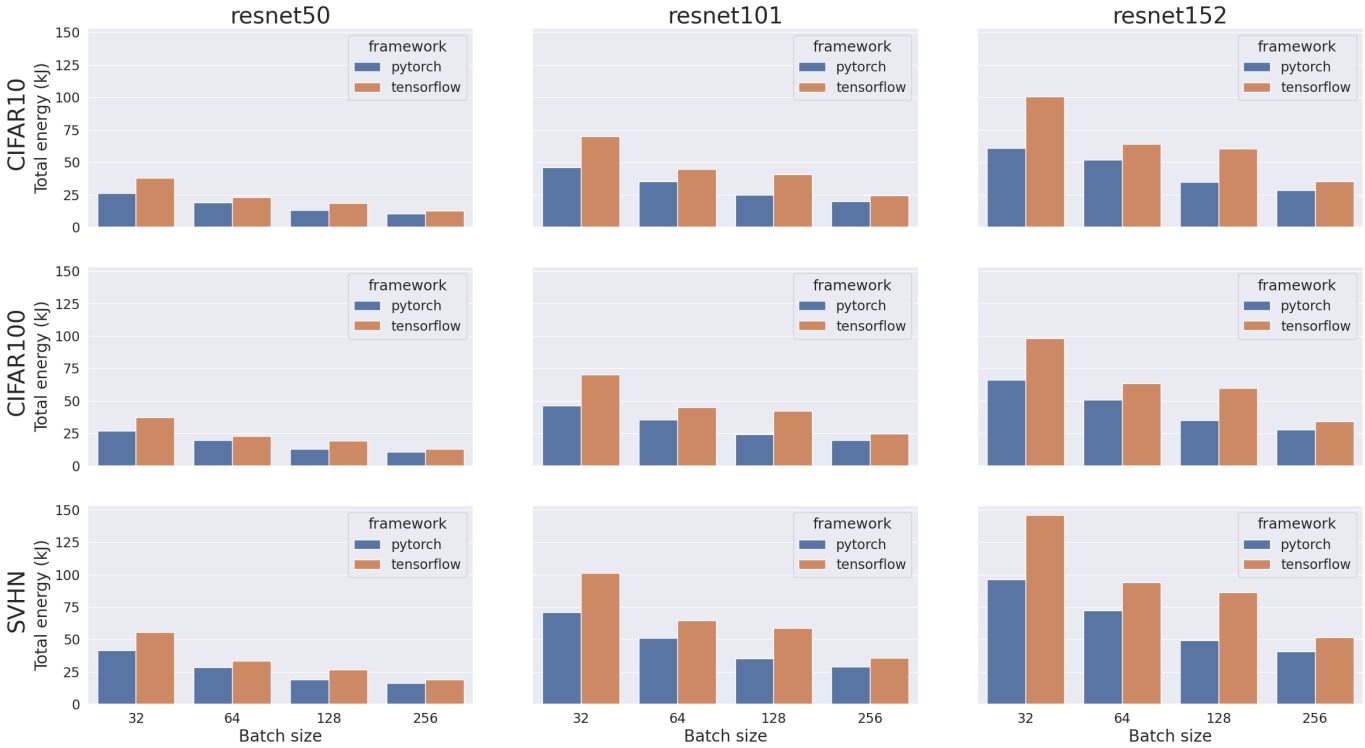

Fig. 2: Total energy (in kJ) used in 10 epochs of training, for each combination of levels. Result of adding up energy used at each epoch (time spent * average power) for all epochs.

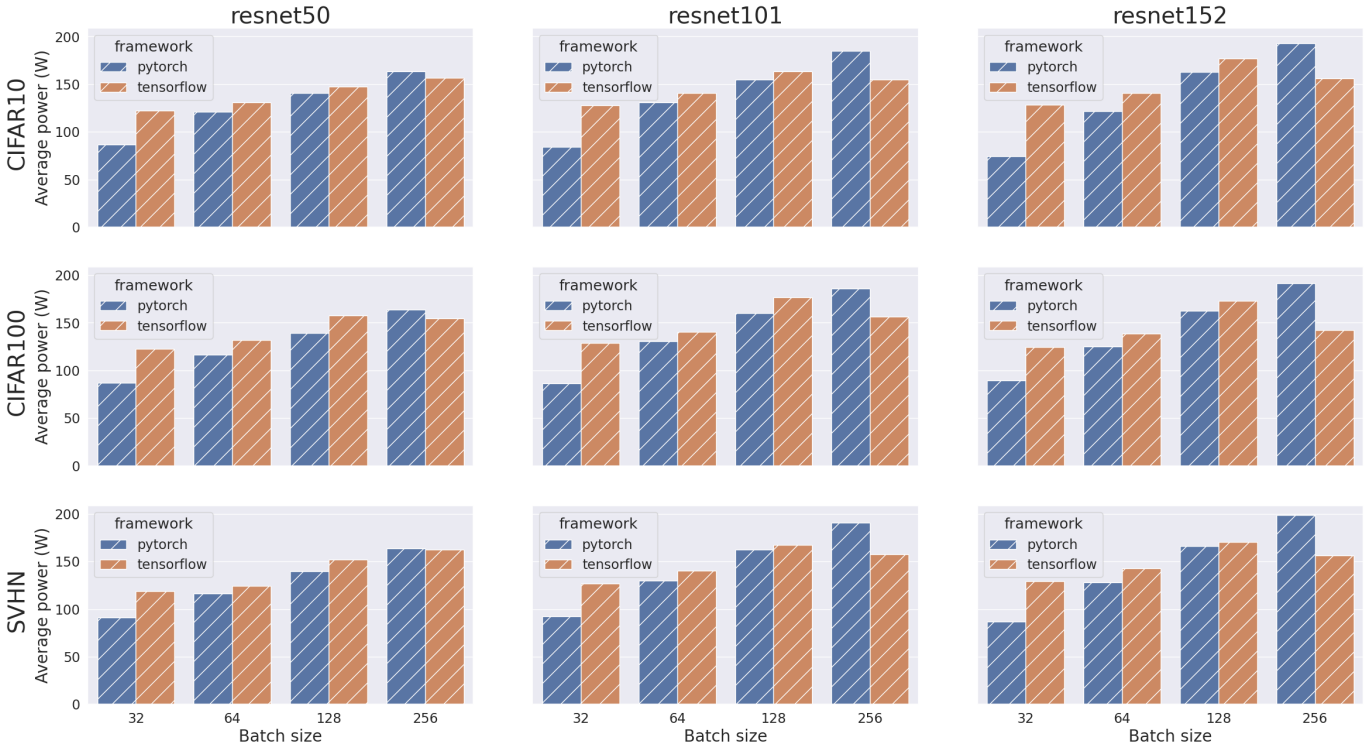

Fig. 3: Average power (in Watts) through 10 epochs of training, for each combination of levels.

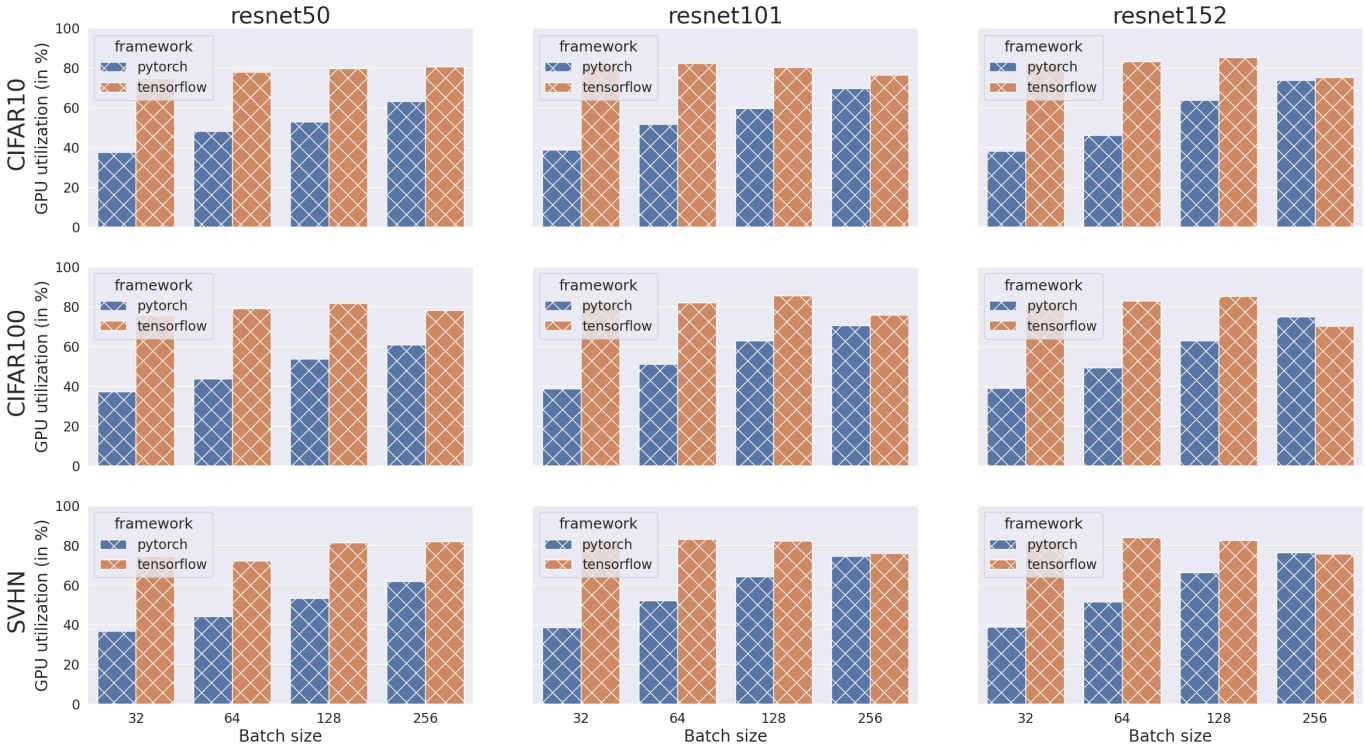

Fig. 4: Average GPU utilization through 10 epochs of training, for each combination of levels.

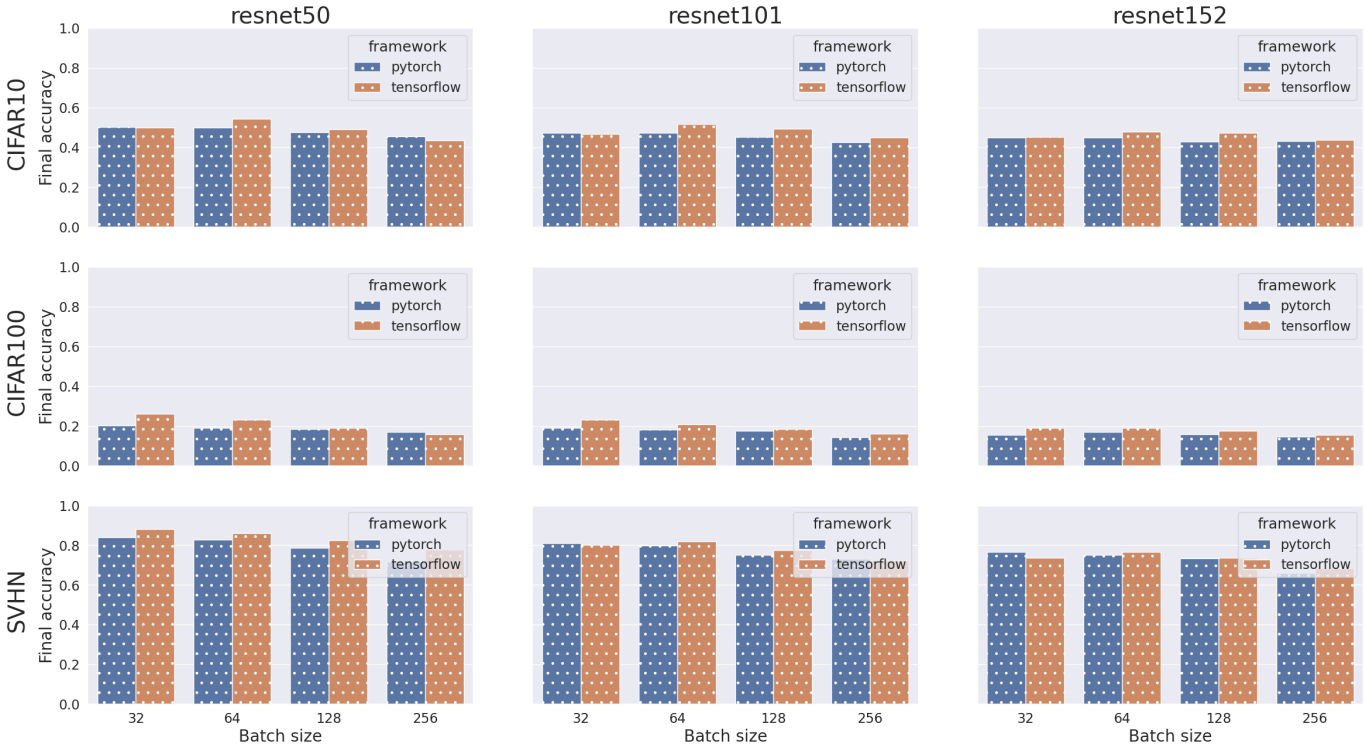

Fig. 5: Final accuracy achieved at the end of the 10-epochs training, for each combination of levels

IV. Concluding remarks

In this project, we conducted an investigation into the power consumption of PyTorch and TensorFlow deep learning frameworks using ResNet architectures. Our goal was to gain insights into the energy efficiency of these frameworks.

Through our experimental methodology, we compared the power consumption of the frameworks by varying the complexity of the ResNet architecture, the complexity of the classification problem, and the batch size. We measured power consumption, total energy usage, GPU utilization, and GPU temperature using Nvidia's nvidia-smi tool.

Based on the results obtained in this project, we can conclude that the energy consumption of deep learning frameworks is highly dependent on the batch size and complexity of the neural network being trained. Larger batch sizes lead to lower energy consumption per batch, while more complex neural networks require more energy to be trained.

It is also evident that PyTorch and TensorFlow have different energy consumption characteristics, with TensorFlow consistently consuming more energy than PyTorch in all scenarios. However, it should be noted that PyTorch finishes the training earlier while having slightly higher average energy usage. This suggests that PyTorch may be more efficient in terms of GPU usage.

In future work, it would be interesting to explore the impact of different hyperparameters, such as learning rate and the optimizer choice, on energy consumption and provide further insights into optimizing the energy efficiency of deep learning frameworks.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 12 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[2] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[3] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning." [Online]. Available: http://ufldl.stanford.edu/housenumbers/

[4] C. S. P. Jorge del Pozo Lerida, Ívar Óli Sigursson, "Computer systems performance," https://github.com/jorgedelpozolerida/ComputerSystemsPerformance, 2023.